[**Planners**](https://journals.sagepub.com/doi/pdf/10.1177/1729881416663663)

*Global Planner (A Star)*

The global planner (GP) is in the asv\_global\_planner package. The implemented global planner is a basic A Star algorithm, with just a post processing algorithm to reduce the number of waypoints. If the waypoints are not calculated by the planner they must be set as parameters in the launch file. It is meant to be used on the processed map with inflated obstacles to guarantee a security margin.

How to add one?

To integrate a new global planner and use it, here are the steps to follow:

-> First add a new .cpp file in asv\_global\_planner/src implementing the class with methods and attributes of the new GP (use the common functions described in global\_planner.cpp). The new class will need to inherit from the class GlobalPlanner (by adding   “public GlobalPlanner”   next to   “class NewPlanner :”  in the .cpp), in particular the following methods will need to be implemented

* void initialize(nav\_msgs::OccupancyGrid \*map)
* asv\_msgs::Path calculate\_waypoints(const double start\_x, const double start\_y, const double arrival\_x, const double arrival\_y)

-> Add in asv\_global\_planner/include/asv\_global\_planner the corresponding .h header, and add the following line at its beginning:

#include "asv\_global\_planner/asv\_global\_planner.h"

-> Find in the CMakeLists.txt the line **61** and add as an argument src/name\_of\_your\_file.cpp to the function:

add\_executable(asv\_global\_planner\_node src/asv\_global\_planner\_node.cpp src/asv\_global\_planner.cpp src/asv\_a\_star.cpp)

-> Go to asv\_global\_planner\_node.h and add this line right after line **15,** with the name of your header file:

#include "asv\_global\_planner/name\_of\_the\_new\_header.h"

-> Then go to asv\_global\_planner\_node.cpp, line **27** and change it to add the constructor of the class of the planner that is wanted:

NewPlanner \*gp = new NewPlanner(\*\*args);

where NewPlanner is the name of the class you created and \*\*args are the arguments of your constructor, if required.

-> Finally, go back to the root of the workspace and launch

catkin\_make

*NB*: this method doesn’t allow to change easily between different global planners because it needs to modify the source code at each change of GP. A better way would be to use the parameter global\_planner in the launch files and use it to parametrize the name of the class of the planner that needs to be used.

*Local Planner (Velocity Obstacle)*

The local planner (LP) is located in the asv\_ctrl\_vo package. The local planner already implemented is a Velocity Obstacle (VO) taking account of the COLREGS. The algorithm is detailed in Stenersen’s thesis. It considers the static obstacles but hardly, and without any security margin. It can be more convenient to feed two different maps to the local and the global planner (when in a channel for example). When it comes to the obstacle ships, the VO considers a security distance defined in asv\_ctrl\_vo/src/asv\_ctrl\_vo\_node.cpp line **149:**

double combined\_radius = RADIUS\_ + it->header.radius;

By default, RADIUS\_ is 10m but the size of the ASV is 8m, so it allows a security distance of only 2m, which is very close and quite dangerous. That is why we worked a lot with a modified version of the planner where we just commented this line and uncommented the previous one (**do not forget to call catkin\_make** **in the main directory of the workspace to compile the changes on a .cpp or .h file**):

double combined\_radius = RADIUS\_\*std::max((v\_ret+0.5),1.0) + it->header.radius;

This distance increases with the relative speed between the two ships, thus resulting in safer and more reasonable security margins. However, it was meant for open seas and this version of the planner might encounter hardships in restricted spaces. It is possible to integrate another local planner, but it would need to modify more in depth the package.

How to add one?

To integrate a new local planner and use it, here are the steps to follow:

-> First add a new .cpp file in asv\_ctrl\_vo/src implementing the class with methods and attributes of the new GP. The functions

* void initialize(std::vector<asv\_msgs::State> \*obstacles, nav\_msgs::OccupancyGrid \*map, ros::NodeHandle nh)
* void update()
* void updateAsvState(const nav\_msgs::Odometry::ConstPtr &msg, const double &u\_d, const double &psi\_d)
* void initializeMarker(visualization\_msgs::Marker \*marker)

will need to be implemented, even if they have to be empty.

-> Add in asv\_ctrl\_vo/include/asv\_ctrl\_vo the corresponding .h header.

-> Find in the CMakeLists.txt the line **86:**

add\_executable(asv\_ctrl\_vo\_node src/asv\_ctrl\_vo\_node.cpp src/asv\_ctrl\_vo.cpp)

and add as an argument src/name\_of\_your\_file.cpp to the function.

-> Then go to asv\_ctrl\_vo/include/asv\_ctrl\_vo/asv\_ctrl\_vo\_node.h, line **59**

VelocityObstacle \*vo\_;

and change it to add a pointer towards an instance of the class you implemented instead of a Velocity Obstacle (named vo\_ too):

NewLocalPlanner \*vo\_;

Then seek in the function void initialize(...)  the argument “VelocityObstacle \*vo,” line **37** and change it to  “NewLocalPlanner \*vo,” .

-> Go to asv\_ctrl\_vo\_node.cpp and do the same last operation on the definition of the function  void initialize(...)  line **79,** and change the creation of a definition of the Velocity Obstacle with the new planner by changing line **24** with:

NewPlanner \*vo = new NewPlanner(\*\*args);

where NewPlanner is the name of the class you created and \*\*args are the arguments of your constructor, if required. Finally, add

#include "asv\_ctrl\_vo/name\_of\_your\_file.h"

At the beginning of the file.

-> Finally, go back to the root of the workspace and launch

catkin\_make

*NB*: Most local planners need a path tracker to work, but some just need an array of waypoints, for the latter, a subscriber to asv/waypoints needs to be added to asv\_ctrl\_vo\_node.cpp at line **25**:

ros::Subscriber og\_sub = n.subscribe("asv/waypoints", 1, VelocityObstacleNode::wpCallback, &vo\_node);

and add a function

void VelocityObstacleNode::wpCallback(const nav\_msgs::Path::ConstPtr &msg)   
{   
    // ...   
}

at the end of the file asv\_ctrl\_vo\_node.cpp. Add it also in the corresponding header. By doing that way, the path tracker node doesn’t need to be run.

*NB2*: this method doesn’t allow to change easily between different local planners because it needs to modify the source code at each change of GP. A better way would be to set parameter local\_planner in the launch files and use it to parametrize the name of the class of the planner that needs to be used. It could then be useful to create a class LocalPlanner from which all the classes of the local planners would inherit.